Zhang, Chunyang
Homework Assignment 1
Instructor Schutfort
June 26, 2015

1) (CLRS) 1.2-2. Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in 64nlgn steps. For which values of n does insertion sort beat merge sort?

   Answer:

   From Figure 1, we can see that these two step functions meet in two points. The first point is between n = 1 and n =2, and the section point happens between n = 43 and n = 44. Therefore, insertion sort beats merge sort when $8n^2 \leq 64nlgn$, which is $2 \leq n \leq 43$.

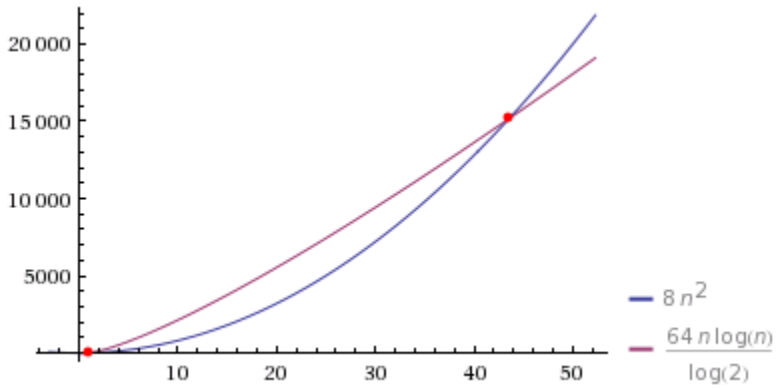

Figure 1: Plot for $8n^2$ and 64nlgn

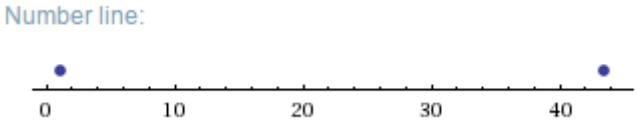

Figure 2: Plot for the two points

2) (CLRS) Problem 1-1 on pages 14-15. Fill in the given table.

   1-1 Comparison of running times
   For each function $f(n)$ and time t in the following table, determine the largest size n of a problem that can be solved in time t, assuming that the algorithm to solve the problem takes $f(n)$ microseconds.

Answer:

In this question we assume 1 month = 30 days and 1 year = 12 month.

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$lg\ n$	2^{10^6}	$2^{6\times10^7}$	$2^{3.6\times10^9}$	$2^{8.64\times10^{10}}$	$2^{2.592\times10^{12}}$	$2^{3.1104\times10^{13}}$	$2^{3.1104\times10^{15}}$
$\sqrt{n}$	10^{12}	3.6×10^{15}	1.296×10^{19}	7.46496×10^{21}	6.718464×10^{24}	9.6758816×10^{26}	9.6758816×10^{30}
n	10^6	6×10^7	3.6×10^9	8.64×10^{10}	2.592×10^{12}	3.1104×10^{13}	3.1104×10^{15}
$n \lg n$	627476	2.80142×10^6	1.33378×10^8	2.75515×10^9	7.18709×10^{10}	7.8709×10^{11}	6.76995×10^{13}
n^2	1000	7746	60000	293939	1609969	5577096	55770960
n^3	100	391	1532	4420	13737	31449	145973
2^n	19	25	31	36	41	44	51
$n!$	9	11	12	13	15	16	17

3) (CLRS) 2.3-3 on page 39. Use mathematical induction to show that when n is an exact power of 2, the solution of the recurrence

$$T(n) = \begin{cases} 2, & if\ n = 2 \\ 2T\left(\frac{n}{2}\right) + n, & if\ n = 2^k, for\ k > 1 \end{cases}$$

is $T(n) = n\ lgn$.

Answer:

When $n = 2, T(n) = T(2) = 2 = 2\ lg2 = n\ lgn$.

When $n = 2^k\ for\ k > 1$,

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$
$$= 2T(2^{k-1}) + 2^k$$
$$= 2 \times 2T(2^{k-2}) + 2^k + 2^k$$
$$= 2^2 \times T(2^{k-2}) + 2 \times 2^k$$
$$= 2^3 \times T(2^{k-3}) + 3 \times 2^k$$
$$= \cdots$$
$$= 2^{k-1} \times T(2) + (k-1) \times 2^k$$
$$= 2^{k-1} \times 2 + (k-1) \times 2^k$$
$$= 2^k + (k-1) \times 2^k$$
$$= k\ 2^k$$

As $n = 2^k, lgn = k$.

Therefore,

$$T(n) = k\ 2^k = n\ lgn.$$

Overall, the solution of the recurrence is $T(n) = n\ lgn$.

4) For each of the following pairs of functions, either f(n) is O(g(n)), f(n) is Ω(g(n)), or f(n) = Θ(g(n)). Determine which relationship is correct and explain.

a. $f(n) = n; g(n) = n^2 - 100n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n}{n^2-100n} = \lim_{n\to\infty} \frac{1}{n-100} = 0$$

$\rightarrow f(n)\ is\ O\big(g(n)\big).$

b. $f(n) = n^{0.25}; g(n) = n^{0.5}$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^{0.25}}{n^{0.5}} = \lim_{n\to\infty} \frac{1}{n^{0.25}} = 0$$

$\rightarrow f(n)\ is\ O\big(g(n)\big).$

c. $f(n) = n;\ g(n) = log^2 n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n}{log^2 n} = \lim_{n\to\infty} \frac{n}{2log\ n} = \lim_{n\to\infty} \frac{1}{2\frac{1}{ln10}\frac{1}{n}} = \lim_{n\to\infty} \frac{ln10}{2} n = \infty$$

$\rightarrow f(n)\ is\ \Omega\big(g(n)\big).$

d. $f(n) = lg\ n;\ g(n) = log^2\ 3n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{lg\ n}{log^2\ 3n} = \lim_{n\to\infty} \frac{lg\ n}{2log\ 3n} = \lim_{n\to\infty} \frac{\frac{1}{n}}{2\frac{1}{3n}} = \lim_{n\to\infty} \frac{3}{2} = \frac{3}{2}$$

$\rightarrow f(n) = \Theta\big(g(n)\big).$

e. $f(n) = log\ n;\ g(n) = lg\ n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{log\ n}{lg\ n} = \lim_{n\to\infty} \frac{ln2}{ln10} = \lim_{n\to\infty} log\ 2 = log\ 2$$

$\rightarrow f(n) = \Theta\big(g(n)\big).$

f. $f(n) = 2^n;\ g(n) = 10n^2$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^n}{10n^2} = \lim_{n\to\infty} \frac{n\ 2^{n-1}}{20n} = \lim_{n\to\infty} \frac{n(n-1)\ 2^{n-2}}{20} = \infty$$

$\rightarrow f(n)\ is\ \Omega\big(g(n)\big).$

g. $f(n) = e^n;\ g(n) = 2^n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{e^n}{2^n} = \lim_{n\to\infty} \left(\frac{e}{2}\right)^n = \infty$$

$$\rightarrow f(n)\ is\ \Omega\big(g(n)\big).$$

h. $f(n) = 2^n;\ g(n) = 2^{n+1}$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^n}{2^{n+1}} = \lim_{n\to\infty} \frac{1}{2} = \frac{1}{2}$$

$$\rightarrow f(n) = \Theta\big(g(n)\big).$$

i. $f(n) = 2^n;\ g(n) = 2^{2^n}$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^n}{2^{2^n}} = \lim_{n\to\infty} \frac{n2^{n-1}}{2^n\, n2^{n-1}\, 2^{2^n-1}} = 0$$

$$\rightarrow f(n)\ is\ O\big(g(n)\big).$$

j. $f(n) = n2^n;\ g(n) = 2^n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n2^n}{2^n} = \lim_{n\to\infty} n = \infty$$

$$\rightarrow f(n)\ is\ \Omega\big(g(n)\big).$$

k. $f(n) = 2^n;\ g(n) = n!$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^n}{n!} = 0$$

$$\rightarrow f(n)\ is\ O\big(g(n)\big).$$

l. $f(n) = (n+1)!;\ g(n) = n!$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{(n+1)!}{n!} = \lim_{n\to\infty} n = \infty$$

$$\rightarrow f(n)\ is\ \Omega\big(g(n)\big).$$

5) Let f_1 and f_2 be asymptotically positive functions. Prove or disprove each of the following conjectures.

a. $f_1(n) = O(f_2(n))$ implies $f_2(n) = O(f_1(n))$.

As $f_1(n) = O(f_2(n))$, then there exists a constant c_1 that

$$0 < f_1(n) \leq c_1 f_2(n), for\ n > n_0$$

In this case we cannot find another constant c_2 that

$$0 < f_2(n) \leq c_2 f_1(n)$$

Therefore, $f_1(n) = O(f_2(n))$ does not imply $f_2(n) = O(f_1(n))$.

b. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$.

As $f_1(n) = O(g_1(n))$, then there exists a constant c_1 that

$$0 < f_1(n) \leq c_1 g_1(n), for\ n > n_1$$

As $f_2(n) = O(g_2(n))$, then there exists a constant c_2 that

$$0 < f_2(n) \leq c_2 g_2(n), for\ n > n_2$$

Then we get

$$0 < f_1(n) \cdot f_2(n) \leq c_1 g_1(n) \cdot c_2 g_2(n), for\ n > max(n_1, n_2)$$

Assume $c_3 = c_1 \cdot c_2$ and $n_3 = max(n_1, n_2)$

$\rightarrow 0 < f_1(n) \cdot f_2(n) \leq c_3\big(g_1(n) \cdot g_2(n)\big), for\ n > n_3.$

Therefore, $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$.

c. $max\ \big(f_1(n), f_2(n)\big) = \Theta\big(f_1(n) + f_2(n)\big)$.

If $max\ (f_1(n), f_2(n)) = \Theta(f_1(n) + f_2(n))$

$$0 < c_1\big(f_1(n) + f_2(n)\big) \leq max\ \big(f_1(n), f_2(n)\big) \leq c_2\big(f_1(n) + f_2(n)\big), for\ n > n_0$$

We know that

$$\frac{1}{2}\big(f_1(n) + f_2(n)\big) \leq \max\big(f_1(n), f_2(n)\big)$$

and

$$\max\big(f_1(n), f_2(n)\big) \leq f_1(n) + f_2(n)$$

Therefore, if we set $c_1 = \frac{1}{2}$ and $c_2 = 1$

Then

$$0 < \frac{1}{2}\big(f_1(n) + f_2(n)\big) \leq max\ \big(f_1(n), f_2(n)\big) \leq f_1(n) + f_2(n), for\ n > n_0$$

$$\rightarrow max\ \big(f_1(n), f_2(n)\big) = \Theta\big(f_1(n) + f_2(n)\big).$$

6) **Fibonacci Numbers:**

The Fibonacci sequence is given by: 0, 1, 1, 2, 3, 5, 8, 13, 21, ….. By definition the Fibonacci sequence starts at 0 and 1 and each subsequent number is the sum of the previous two. In mathematical terms, the sequence F_n of Fibonacci number is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2} \; with \; F_0 = 0 \; and \; F_1 = 1$$

An algorithm for calculating the n^{th} Fibonacci number can be implemented either recursively or iteratively.

a) Implement both recursive and iterative algorithms to calculate Fibonacci Numbers in the programming language of your choice. Provide a copy of your code with your written homework assignment.

The C code is written as follows:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int recursivefib(int n)
{
	if (n == 0)
		return 0;
	else if (n == 1)
		return 1;
	else
		return (recursivefib(n - 1) + recursivefib(n - 2));
}

long iterativefib(long n)
{
	long fib = 0, a = 1, t = 0;
	for (long k = 0; k < n; k++)
	{
		t = fib + a;
		a = fib;
		fib = t;
	}
	return fib;
}

int main()
{
	clock_t timer;

	/*recursive algorithm*/
	int n = 30;
	int recursivesum;
	timer = clock();
	recursivesum = recursivefib(n);
	timer = clock() - timer;
```

```
        printf("The %dth number of this Fibonacci Numbers using recursive algorithm
is: %d\n", n, recursivesum);
        printf("recursive algorithm for n = %d ran in %f seconds\n\n", n, (float)timer /
(float)CLOCKS_PER_SEC);

        /*iterative algorithm*/
        long n = 10000000;
        long iterativesum;
        timer = clock();
        iterativesum = iterativefib(n);
        timer = clock() - timer;
        printf("The %ldth number of this Fibonacci Numbers using iterative algorithm
is: %ld\n", n, iterativesum);
        printf("iterative algorithm for n = %ld ran in %f seconds\n\n", n, (float)timer /
(float)CLOCKS_PER_SEC);
    }
```

b) Use the system clock to record the running times of each algorithm for n = 5, 10, 15, 20, 30, 50, 100, 1000, 2000, 5000, 10,000. You may need to modify the values of n if an algorithm runs too fast or too slow.

n for recursive algorithm	25	30	35	40	45	50
Running time (s)	0	0.01	0.17	1.97	21.92	242.360001

n for iterative algorithm ($\times 10^6$)	1	10	100	1000	2000	5000	10000	20000	50000
Running time (s)	0	0.03	0.28	2.78	5.56	13.89	27.80 9999	55.63 0001	139.0 09995

c) Plot the running time data you collected on graphs with n on the x-axis and time on the y-axis. What type of function (curve) best fits each data set? Discuss the differences in the running times of each algorithm.

The matlab code is written as follows:

```
n = [0  25  30  35  40  45  50];
t = [0  0   0.01    0.17    1.97    21.92   242.360001];
figure(1);
plot(n,t);
xlabel('n (recursive algorithm)');
ylabel('Running time (s)');

x = [0  1   10  100 1000    2000    5000    10000   20000   50000].*1e6;
y = [0  0   0.03    0.28    2.78    5.56    13.89   27.809999   55.630001
139.009995];
figure(2);
plot(x,y);
xlabel('n (iterative algorithm)');
ylabel('Running time (s)');
```

Then we got two plots as follows:

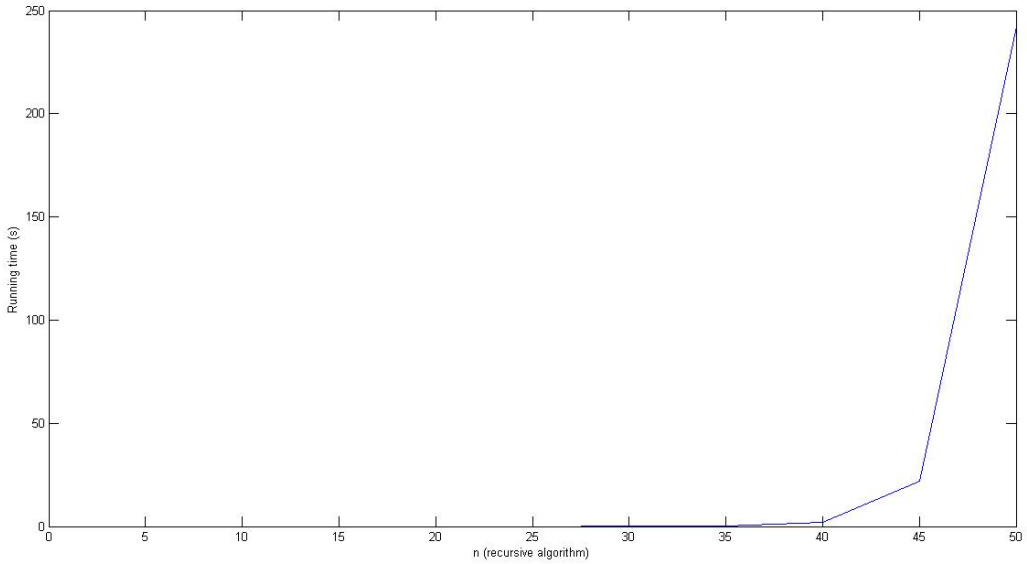

Figure 3: Plot for recursive algorithm

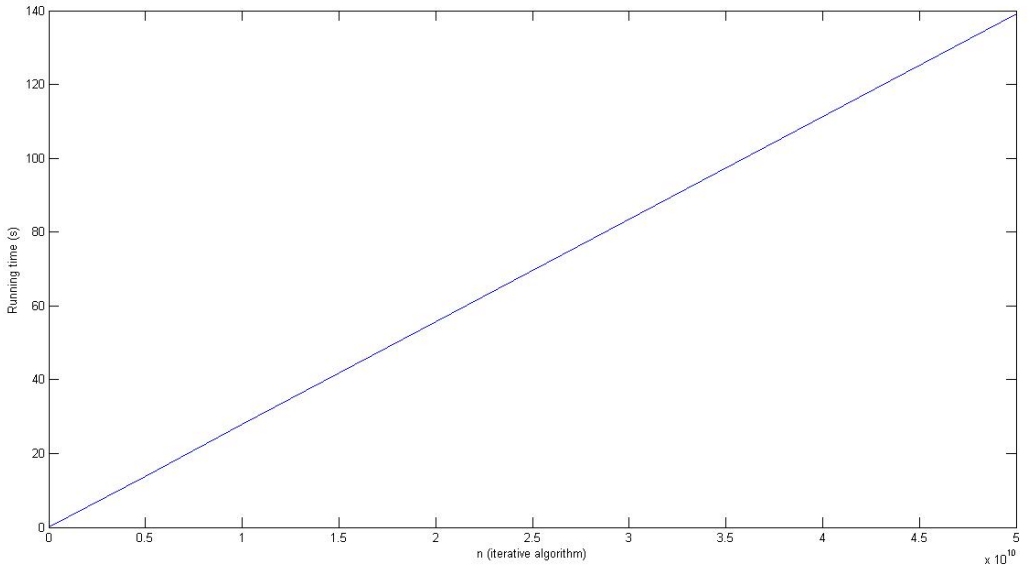

Figure 4: Plot for iterative algorithm

From these data and two graphs, we can easily tell that the recursive algorithm is exponential function, while the iterative algorithm is linear function. Therefore, the recursive algorithm is $\Theta(x^n)$ $(x > 1)$ while the iterative algorithm is $\Theta(n)$. In this case, the iterative algorithm beats the recursive algorithm.

7) Describe a $\Theta(n\ lgn)$- time algorithm that, given a set S of n integers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x. Give an example.

Answer:

First, insert every integer of set S into a binary search tree ($\Theta(n\ lgn)$), and use in-order tree traversal to sort these values into an array ($\Theta(n\ lgn)$). Then, pick the first value a from the sorted list and calculate $y = x - a$. Use binary search algorithm ($\Theta(lgn)$) to find y from the rest of the sorted list. If y cannot be found in the sorted list, then pick the second value as a and calculate $y = x - a$ again. Look for y again and redo the algorithm if it cannot be found until y is found in the sorted list. The algorithm for picking the value overall is $\Theta(n)$, therefore the whole algorithm for searching is $\Theta(n\ lgn)$ for the worst case. On the whole, the time for the entire algorithm is $\Theta(n\ lgn)$.

For instance, given the set S = [4, 6, 8, 0, 10, 2], and x = 12.

The sorted list of S is [0, 2, 4, 6, 8, 10]. First pick $a = 0$, so that $y = x - a = 12$.

Use binary search algorithm to find 12, and it cannot be found, therefore we pick $a = 2$, and $y = x - a = 10$. Use binary search again and we can find 10 in the list. In this case, the algorithm is completed.

Extra Credit:

$$f(n) = \sum_{i=0}^{k} b_i n^i \in \Theta(n^k)$$

For $f(n) \in \Theta(n^k)$, we get that $0 < c_1 n^k \leq f(n) \leq c_2 n^k, for\ n > n_0\ and\ c_1, c_2 > 0$.

Because $0 < b_k n^k < \sum_{i=0}^{k} b_i n^i < \sum_{i=0}^{k} b_i n^k, for\ n \geq 1$,

then we can assume that

when $c_1 = b_k\ and\ c_2 = \sum_{i=0}^{k} b_i$,

$$0 < c_1 n^k \leq f(n) \leq c_2 n^k, for\ n > 1$$

Therefore, $f(n) \in \Theta(n^k)$.